Strong WORM

Radu Sion*
Network Security and Applied Cryptography Lab
Computer Science, Stony Brook University
sion@cs.stonybrook.edu

Abstract

We introduce a Write-Once Read-Many (WORM) storage system providing strong assurances of data retention and compliant migration, by leveraging trusted secure hardware in close data proximity. This is important because existing compliance storage products and research prototypes are fundamentally vulnerable to faulty or malicious behavior, as they rely on simple enforcement primitives ill-suited for their threat model. This is hard because tamper-proof processing elements are significantly constrained in both computation ability and memory capacity – as heat dissipation concerns under tamper-resistant requirements limit their maximum allowable spatial gate-density. We achieve efficiency by (i) ensuring the secure hardware is accessed sparsely, minimizing the associated overhead for expected transaction loads, and (ii) using adaptive overhead-amortized constructs to enforce WORM semantics at the throughput rate of the storage servers ordinary processors during burst periods. With a single secure co-processor, on single-CPU commodity x86 hardware, our architecture can support over 2500 transactions per second.

1 Introduction

Over 10,000 regulations govern the management of information in the US alone [25], in financial, life sciences, health-care industries and the government. These regulations impose a wide range of regulatory policies, ranging from information life-cycle management (e.g., mandatory data retention and deletion) to audit trails and storage confidentiality. Examples include the Gramm-Leach-Bliley Act [19], the Health Insurance Portability and Accountability Act [30] (HIPAA), the Federal Information Security Management Act [31], the Sarbanes-Oxley Act [32], the Securities and Exchange Commission rule 17a-4 [29], the DOD Records Management Program under directive 5015.2 [26], the Food and Drug Administration 21 CFR Part 11 [28], and the Family Educational Rights and Privacy Act [27].

A recurrent theme to be found in these regulations is the need for regulatory-compliant storage as an underpinning to deliver Write Once Read Many (WORM) assurances, essential for enforcing long-term data retention and life-cycle policies. Main requirements are:

Guaranteed Retention. One main goal of compliance storage is to support WORM semantics: once written, data cannot be undetectably altered or deleted before the end of their regulation-mandated life span, even with physical access to the store.

Secure Deletion. Once a record has reached the end of its lifespan, it can (and in some cases must) be deleted. Deleted records should not be recoverable even with unrestricted access to the underlying storage medium; moreover, deletion should leave no hints of their existence at the storage server.

Compliant Migration. Retention periods are measured in years, e.g., intelligence information, educational and health records have retention periods of over 20 years. Accordingly, *compliant data migration* mechanisms are required to transfer information from obsolete to new storage media while preserving the associated security assurances.

A common thread running through many of these regulations is the perception of powerful insiders as the primary adversary. These adversaries have superuser powers coupled with full access to the storage system hardware. This corresponds to the perception that much recent corporate malfeasance has been at the behest of CEOs and CFOs, who also have the power to order the destruction or alteration of incriminating records. Since the visible alteration or destruction of records is tantamount to an admission of guilt in the context of litigation, a successful adversary must perform their misdeeds *undetectably*.

Major storage vendors have responded by offering compliance storage and WORM products, including IBM [14], and EMC [7]. Unfortunately, these products and research prototypes do not satisfy the requirements outlined above.

*The author is supported partly by the NSF through awards CT CNS-0627554, CT CNS-0716608 and CRI CNS-0708025. The author also wishes to thank Motorola Labs, IBM Research, CEWIT, and the Stony Brook VP for Research.

There are fundamental vulnerabilities to faulty or malicious behavior, because of the reliance on simple enforcement primitives such as software and/or simple hardware device-hosted on/off switches – ill-suited to the target (insider) threat model. In practice, these first-generation mechanisms allow an insider using off-the-shelf resources to replicate illicitly modified versions of data onto seemingly-identical storage units without detection.

More generally, the design of compliance storage is extremely challenging due to the conflict between security, cost-effectiveness, and efficiency. To defend against insiders, we need processing components that are both tamper-resistant and *active*, such as general-purpose trustworthy hardware. By offering the ability to run logic within a secured enclosure, such devices allow fundamentally new paradigms of trust. Trust chains spanning untrusted and possibly hostile environments can now be built. The trusted hardware will run certified logic; close proximity to data coupled with tamper-resistance guarantees allow an optimal balancing and partial decoupling of the efficiency/security trade-off. Assurances can now be both efficient and secure.

However, practical limitations of trusted devices pose significant challenges in achieving sound compliance. Heat dissipation concerns under tamper-resistant requirements limit the maximum allowable spatial gate-density. Thus general-purpose secure coprocessors (SCPUs) are significantly constrained in both computation ability and memory capacity, being up to one order of magnitude slower than host CPUs. This mandates careful consideration in achieving efficient protocols. Straight-forward implementations of the full processing logic *inside* SCPUs are bound to fail in practice simply due to lack of performance. The server's main CPUs will remain starkly under-utilized and the entire cost-proposition of having fast untrusted main CPUs and expensive slower secured CPUs will be defeated.

2 Model

2.1 Economics and Threat Model.

In the threat model for compliance storage, a legitimate user Alice creates and stores a record (e.g., b_2) onto WORM storage. Later, b_2's existence is regretted and Alice (acting in effect as a malicious "Mallory") will do everything she can to prevent Bob (e.g., federal investigators) from accessing b_2 or inferring its existence. The main purpose of WORM is to defend against such an Mallory. Moreover, as she may have superuser powers, and direct physical access to the hardware, we cannot rely on conventional file/storage system access control mechanisms or data outsourcing techniques to ensure that records are modifiable only in compliance with regulation.

To prevent physical attacks, strong tamper-resistant and reactive hardware is required. Moreover, due to the data intensive nature of the application, such hardware should allow the execution of WORM logic inside its trusted enclosure. This is one of the reasons why passive hardware such as specified by the Trusted Platform Module [3] specifications of the Trusted Computing Group can not be deployed.

Regarding the requirements of *secure deletion* (at the end of mandated retention periods) that can often be found in regulation, we note that Alice has always the natural choice of "remembering" records past their regulation-mandated retention period, e.g., in non-regulated outside storage. Thus, in the WORM adversarial model the focus is mainly on preventing Alice from "rewriting" history, rather than "remembering" it. Additionally, we prevent the rushed removal of records before their retention periods.

2.2 Deployment

To enforce strong WORM semantics, we are augmenting a traditional storage cluster with a set of trusted FIPS 140-2 Level 4 certified hardware components as main points of processing trust and tamper-proof assurances. Our architecture employs trusted general-purpose hardware such as the IBM 4758 PCI and the newer IBM 4764 PCI-X [2] cryptographic coprocessors. The IBM 4764 is a PowerPC-based board and runs embedded Linux. It can be custom programmed to run arbitrary code. Moreover, it is compatible with the IBM Common Cryptographic Architecture (CCA) API [1]. The CCA implements common cryptographic services such as random number generation, key management, digital signatures, and encryption (DES/3DES,RSA). If physically attacked, the device destroys internal state (in a process powered by internal long-term batteries) and shuts down in accordance with the FIPS 140-2 certification.

Note on timestamps: At various points in this paper timestamps generated by the SCPU are deployed to assert the freshness of integrity constructs. In this context it is important to note that the considered SCPUs maintain internal, accurate clocks protected by their tamper-proof enclosure.

2.3 Cryptographic Tools

We assume readers are familiar with semantically secure (IND-CPA) encryption and signature mechanisms [10] and *cryptographic hashes* [16]. We consider ideal, collision-free hashes and strongly un-forgeable signatures. *Merkle (hash) trees* [17] enable the authentication of item sets by using only a small amount of information. As suggested in the data outsourcing literature (where the adversary is an outsider), Merkle trees are a useful tool in guaranteeing data integrity. However, in a compliance storage environment, where new records are constantly being added to the store,

Merkle tree updates ($O(\log n)$ costs) can be a performance bottleneck. Our solution will overcome this by deploying a simple yet efficient range authentication technique relying on the certifying entire "windows" of allocated records (with $O(1)$ update costs).

3 Related Work

In this section we briefly discuss existing WORM mechanisms as well as a set of related research areas such as encryption and versioning file systems. Please refer to the full version of the paper for more details.

Tape-based WORM. Tape-based WORM assurances are provided under the assumption that only approved tape-readers are deployed. However, given the nature of magnetic tape, an attacker can easily dismantle the plastic tape enclosure and access underlying data on a different customized reader.

Optical-disk WORM. Optical disk WORM guarantees rely on irreversible physical primitive write effects to ensure the inability to alter existing content. However, this also makes optical disks unsuited for scenarios with variable retention periods. Moreover, slow performance, small data capacity, simple data replication attacks, and the inability to fine-tune secure deletion granularity limit their use in compliance scenarios.

Hard disk-based WORM. Magnetic disks currently offer better overall cost and performance than optical or tape recording. Thus all recently-introduced WORM storage devices are built atop conventional rewritable magnetic disks, with write-once semantics enforced through software ("soft-WORM"). The EMC Centera Compliance Edition [7] is representative of such soft-WORM offerings. However its software-only nature renders it vulnerable to simple insider software and/or physical direct disk-access attacks. Data integrity can be easily compromised. The same problems arise with other vendors such as IBM [14].

Worth noting is the work by Huang et.al. [13], which introduces **"content immutable storage"**, a mostly soft-WORM storage system that must satisfy the following properties: "1) offer overwrite protection that is secure even against inside attacks; 2) efficiently support index mechanisms; 3) allow records to be properly disposed of after they have expired; and 4) be low cost yet reliable" [13].

Unfortunately, the proposed mechanisms stop short of delivering in realistic adversarial settings. Insiders with super-user powers and physical access can simply open the magnetic drive enclosures and alter the underlying media while remaining un-detected. Attempts to prevent this by storing checksum data at locations logically un-addressable from user-land are bound to fail for an insider with full super-user privileges and physical access to the device.

Proposing to provide tamper-proof disks to solve this is a worthy initiative, yet very costly and difficult to attain in practice exactly due to the same reasons why SCPUs are expensive, namely the inability to properly dissipate heat through a tamper-proof enclosure. Moreover, even if such devices would be designed, as in any normal system operation, the associated magnetic media MTBFs will lead to several failed disks per day. If each disk is indeed designed for tamper-proofness, its cost will be comparable to the cost of SCPUs – where a major part of the cost is the certification process and the tamper-proof enclosure – leading to daily maintenance costs of hundreds of thousands of dollars.

Versioning File Systems. Versioning file systems [20, 23] trace and store file changes, making user actions revocable. With digital audit trails for versioning file systems [20], the user publishes a small amount of data to a third party, thus committing to a version history. The published data can be later used by an auditor to verify the contents of the file system. While presenting some conceptual similarities to our work, such approaches suffer from (i) significant privacy concerns – few companies would trust third parties with their internal transaction data, (ii) a set of often impractical assumptions with grave scalability problems – the existence of a globally trusted third party that can support trillions of transactions per second – from the millions of companies registered in the U.S. alone, and (iii) performance drawbacks – network-limited bandwidth and high latency.

Provenance-Aware Storage. A Provenance-Aware Storage System (PASS) [12] automatically collects and maintains "provenance" of documents, which is defined as the complete execution history that produced them.

Integrity-Assured Storage. File systems such as I^{3}FS [15] and GFS [9] perform online real-time integrity verification for file systems. Venti [22] is an archival storage system that performs integrity assurance on read-only data.

Encrypted Storage. Many researchers have proposed file-system-level support for encryption [5, 11, 18, 33]. If native disk support for encryption is not available, these software approaches can fill that role, though at slower speeds. Some of these systems also support data integrity, yet not under the WORM adversarial model.

4 Architecture

4.1 Overview

We achieve strongly compliant storage in adversarial settings by deploying tamper-resistant, general-purpose trustworthy hardware, running certified logic at the server site. As heat-dissipation concerns greatly limit the performance of such tamper-resistant secure processors (SCPUs), it is essential to design a protocol stack with minimal impact on

cost and efficiency. Specifically, we ensure the access to secure hardware is sparse, to minimize the SCPU overhead for expected transaction loads. We deploy special deferred-signature schemes to enforce WORM semantics at the target throughput rate of the storage servers main processors.

Design Vision. We believe it is important for the record-level WORM layer to be simple and efficient. Thus, the focus of this paper is on record-level logic only. Specifically, we *do not discuss name spaces, indexing or content addressing* here, as these do not constitute the main thrust. The mechanisms introduced here can be *layered at arbitrary points in a storage stack*. In most implementations we expect them to be placed either inside a file system (records being files, VRDs acting effectively as file descriptors), or inside a block-level storage device interface (e.g., in embedded scenarios without namespaces or indexing constraints).

Small Trusted Computing Base. The main intuition behind our design is based on the use of the SCPU as a trusted witness to any regulated data updates (i.e., writes and deletions). As such, the SCPU is involved in *updates* only but not in *reads*, thus minimizing the overhead for a query load dominated by read queries. The SCPU witnessing is designed to allow the main CPU to solely handle reads while providing full WORM assurances to clients (who only need to trust the SCPU). Specifically, upon reading a regulated data block, clients are offered SCPU-certified assurances that (i) the block was not tampered with, if the read is successful, or — if the read fails, either (ii) the block was deleted according to its retention policy, or (iii) it never existed in this store.

No Hash-Tree Authentication. To escape the $O(logn)$ per update cost of the straight-forward choice of deploying Merkle trees in data authentication, we introduce a novel mechanism with identical assurances but constant cost per update. To achieve this, we will label data blocks with *monotonically increasing* consecutive serial numbers and then introduce a concept of sliding "windows" which can now be authenticated with constant costs by only signing their boundaries, due to their (consecutive) monotonicity (vs. going up the Merkle tree in $O(logn)$). In doing so we lose some Merkle-tree expressiveness not required here, namely the ability to handle arbitrary (non-numeric) labels.

Peak Performance. During high system-load periods, to further increase throughput we temporarily defer *expensive* witnessing operations (e.g., 1024-bit signatures) with less expensive *short-term secure* variants (e.g., on 512-bit). The short-term security is adaptive and ensures that the system can strengthen these weaker constructs later, during decreased load periods – but within their security lifetime. Thus, the protocols adaptively amortize costs over time and gracefully handle high-load update bursts.

Field	Description
SN	A system-wide unique 64-80 bit serial number.
attr	WORM-related attributes, including creation time, retention period, applicable regulation policy, shredding algorithm, litigation hold, f_flag, MAC, DAC attributes
RDL	The **R**ecord **D**escriptor **L**ist – a list of physical data record descriptors corresponding to the current VR $\{RD_1,\ RD_2,\ ...\}$.
metasig	SCPU signature on (**SN**, **attr**): $S_s\ (SN,\ attr)$.
datasig	SCPU signature on **SN** and a chained hash (or other incremental secure hashing [4, 6]) of the data records : $S_s\ (SN,\ Hash\ (data))$.

Table 1. Outline of a VRD.

4.2 Building Blocks

In the following we detail the above intuitions and main solution building blocks. We define:

1. **Data record.** A data item governed by storage-specific regulation. Data records are application specific and can be files, inodes, database tuples. Records are identified by descriptors (RDs).
2. **Virtual record (VR).** A VR basically groups a collection of records that fall under the same regulation specific requirements (e.g., identical retention period) and need to be handled together. VRs are allowed to overlap, and records can be part of multiple different VRs (being referenced through different descriptors). This enables a greater flexibility and increased expressiveness for retention policies, while allowing repeatedly stored objects (such as popular email attachments) to potentially be stored only once.
3. **Virtual record descriptor (VRD).** A unique, securely issued identifier for a VR. Its structure is outlined in Table 1. A VRD is uniquely identified by a securely issued system-wide serial number (`SN`), and contains various retention-policy related attributes (`attr`), a list of physical data record descriptors (`RDL`) for the associated VR data records, and two trusted signatures (`metasig` and `datasig`) issued by the SCPU, authenticating the `attr` and `RDL` fields.
4. **Virtual record descriptor table (VRDT).** A table of VRDs indexed by their corresponding SNs maintained by the main (un-trusted) CPU on disk.

4.2.1 The VRDT structure

The untrusted main CPU maintains (on disk) a table of VRDs (VRDT) indexed by their corresponding serial num-

bers. These serial numbers are issued by the SCPU at each update. The SCPU securely maintains two private signature keys, s and d respectively, that can be verified by WORM data clients. Their corresponding public key certificates – signed by a regulatory or general purpose certificate authority – are made available to clients by the main CPU.

The SCPU deploys s for the `metasig` and `datasig` signatures in the VRD and d to provide deletion "proofs" that the main CPU can present to clients later requesting the deleted records. Specifically, when the retention period for a record v expires, in the absence of litigation holds, its corresponding entry in the VRDT is replaced by $S_d(v.SN)$. A VR v can be in one of two mutually-exclusive states:

1. *active*: data records and attribute integrity is enforced by the `metasig` $= S_s(SN, attr)$ and `datasig` $= S_s(SN, Hash(data))$ signatures, or

2. *expired*: with the associated "deletion proof" signature $S_d(v.SN)$ present in the VRDT.

Thus, the VRDT entries contain either the VRD for active VRs, or the signed serial number for records whose retention periods have expired.

Window Management. Serial number issuing and VRDT management are designed to minimize the VRDT-related storage. The main idea is to use a sliding window mechanism through which previously expired records' deletion proofs can be safely expelled and replaced with a securely signed lower window bound. To handle the fact that while some of retention expirations are likely to occur in the order of insertion, this is unlikely to hold for all records, an additional data structure that controls record expiration will be required and discussed later.

Specifically, we denote the lowest serial number among all the still *active* VRs (whose retention period has not passed and/or have a litigation hold) as SN_{base}. Let $SN_{current}$ be the highest currently assigned SN. Then the window defined by these two values contain all the active VRs (and possibly a few already expired ones). Any deletion proofs outside of this window are not of WORM-interest any more, and can be securely discarded. Now the main CPU can convince clients that any of the records outside of the current windows have been rightfully deleted (or have not been allocated yet) by simply providing $S_s(SN_{base})$ and $S_s(SN_{current})$ as proofs.

In order to prevent the main CPU using old $S_s(SN_{current})$ values to maliciously ignore recently added records, one of two mechanisms need to be applied: (i) upon each access, the client contacts the SCPU directly to retrieve the current $S_s(SN_{current})$, or (ii) $S_s(SN_{current})$ will also contain a timestamp and the client will not accept values older than a few minutes – and the SCPU will update the signature timestamps on disk every few minutes (even in the absence of data updates). In general cases, we believe (ii) should be chosen for the following reasons: in a busy data store, the staleness of the timestamp on $S_s(SN_{current})$ is not an issue, due to the continuously occurring updates; on the other hand, in an idle system, the small overhead of a signature every few minutes does not impact the overall throughput.

Naturally, to reduce storage requirements, a similar technique can be applied further for different expiration behaviors. Specifically, if records do not expire in the order of their insertion – likely if the same store is used with data governed by different regulations – we can define the following convention: the main CPU will be allowed to replace any contiguous VRDT segment of 3 or more expired VRs with SCPU signatures on the *upper* and *lower* bounds of this deletion "window" defined by the expired SNs segment. This in effect enables multiple active "windows", linked by these signed lower/upper bound pairs for the deleted "windows". Since the trusted signatures result in additional SCPU overhead, we can deploy these storage reduction techniques during idle periods.

It is important to note that the upper and lower deletion window bounds will need to be correlated, e.g., by associating the same unique random window ID to both (e.g., inside the signature envelope). This correlation prevents the main CPU to combine two unrelated window bounds and thus in effect construct arbitrary windows. Also, in order to avoid replay attacks of old $S_s(SN_{base})$ signatures (e.g., Mallory does not want to properly expire records) they will include expiration times. Moreover, such replays would not achieve much, as the clients have always the option to re-verify the correct record retention upon read.

4.2.2 WORM Operations

Write. In a **write**, the following operations are executed. The main CPU writes the actual data to the disk, and messages the SCPU with the resulting RDs and the corresponding attributes (such as regulation policy, retention period and shredding method parameters). *Data records and their RD descriptors are implementation specific and can be inodes, file descriptors, or database tuples*.

The SCPU increments the current serial number counter to allocate a SN for this new VR and then generates its `metasig` and `datasig` signatures. To create `datasig` the SCPU is required to read the data associated with the stored record. It is possible to reduce this overhead at burst-periods under a slightly weaker security model in which the main CPU will be trusted to provide `datasig`'s hash which will be verified later during idle times. The evaluation in section 5 considers both models.

Next, the main CPU creates a VRD, associates it with the specified attributes, as well as `datasig` and `metasig`,

both provided by the SCPU. The VRD is then written by the main CPU to the VRDT maintained in unsecured storage.

Read. To perform a read, a record handle (i.e., the SN) needs to be provided to the WORM layer. It is important to note that, as discussed in section 4.1, the associated data indexing and SN management mechanisms are intently not discussed here.

A client's **read** operation only requires main CPU cycles. This is important, as query loads are expected to be often mostly read-only. If a read of a VR v is disallowed on grounds of expired retention, the main CPU will then either provide $S_d(v.SN)$ (proof of deletion), or prove that the serial number of v is less than SN_{base} (thus rightfully deleted) by providing $S_s(SN_{base})$. Similarly, in the case of the multiple "windows" solution (see section 4.2.1), the main CPU will need to provide a SCPU-signed lower and upper bounds for the window of expired SNs that contains v, as proof of v's deletion.

In a successful read the client receives a VRD and the data. It then has the option of verifying the SCPU `datasig` and `metasig` signatures[1]. If the signatures do not match, the client is assured that the data (or the corresponding VRD) has been prepensely modified or deleted. This is so because the (consecutive) monotonicity of the serial numbers allow efficient discovery of discrepancies.

Record Expiration. Record **expiration** and their subsequent deletion is controlled by a specialized Retention Monitor (RM) daemon running inside the SCPU. To amortize linear scans of the VRDT while ensuring timely deletion of records, the SCPU maintains a sorted (on expiration times) list of serial numbers (VEXP), subject to secure storage space. The VEXP is updated during light load periods (e.g., night-time). As common retention rates are of the order of years, we expect this to not add any additional overhead in practice. The VEXP is deployed by the SCPU to enable efficient and timely deletion of records. To this end, the RM is designed to wake up according to the next expiring entry in VEXP and invokes a **delete** operation on this entry. It then sets a wake-up alarm for the next expiration time and performs a sleep operation to minimize the SCPU processing load. If a new record with an earlier expiration time is written in the meantime, the SCPU resets the alarm timer to this new expiration time and updates the VEXP accordingly.

To **delete** a record v, the SCPU first invokes the associated storage media-related data shredding algorithms for v (not discussed). It then provides the main CPU with $S_d(v.SN)$, the proof of v's rightful deletion of, which will replace v's entry in the VRDT. The main CPU can then show this signature as proof of rightful deletion to clients.

[1]The client must have access to the appropriate SCPU public key certificates (the server can provide them), and have access to a (roughly) synchronized time server.

Litigation. Often, records involved in ongoing litigation proceedings will reside in active WORM repositories. A court can then mandate a litigation hold to be placed on such active records, which in effect will prevent their deletion even if mandated retention periods have expired – such records cannot be deleted until litigation release. This is achieved through **lit_hold** and **lit_release** operations both of which will alter the `attr` field to set a litigation held flag together with an associated timeout of the hold. This process is going to be performed by the SCPU, who will subsequently update also `metasig`.

Litigation holds can be set only by authorized parties identified with appropriate credentials. In their simplest form, these credentials can be instantiated as a verifiable regulation-authority signature on the record's SN, the current time stamp $C = S_{reg}(SN, current\_time)$ (and an optional litigation identifier). This signature can be stored as part of the `attr` field, e.g., to allow the removal of the hold by the same authority only (or other similar semantics). This will be achieved by invoking **lit_release**.

4.3 Optimizations: Deferring Strength

A novel throughput-optimizing method we deploy is to temporarily defer *expensive* witnessing operations (e.g., 1024-bit signatures) with less expensive temporary *short-term secure* variants (e.g., on 512-bit). This is particularly important during update burst periods. The short-lived signatures will then be strengthened (e.g., by resigning with strong permanent keys) during decreased load periods – *but within their security lifetime*. In effect this optimization amortizes SCPU loads over time and thus gracefully handles high-load update bursts.

We use 512-bit RSA signatures as a reference security lower-bound baseline. 512-bit composites could be factored with several hundred computers in about 2 months around year 2000 [24]. However, despite numerous efforts no significant progress has been made [8] beyond Number Field Sieve (NFS) [21] techniques.

To be on the safe side, we assume that today, 512-bit composites can resist no more than a few *tens of minutes* (e.g., 60-180 mins) to factoring attempts by Alice, who may want to do so in order to alter the `metasig` and `datasig` fields. In the WORM adversarial model however, this can only rarely be of concern, as Alice is unlikely to regret record storage and also succeed in breaking the signatures within such short time-frames.

The intuition then is to deploy fast shorter-lived signatures during burst periods to support high transaction rates. To achieve an adaptive behavior, optimally balancing the performance-security trade-off, we need to determine the maximum signature strength we can afford (e.g., bit-length of key) for a given throughput update rate. The main idea

Function	Context	IBM 4764	P4 @ 3.4Ghz
RSA sig.	512 bits	4200/s (est.)	1315/s
	1024 bits	848/s	261/s
	2048 bits	316-470/s	43/s
SHA-1	1KB blk.	1.42 MB/s	80 MB/s
	64 KB blk.	18.6 MB/s	120+ MB/s
DMA xfer	end-to-end	75-90 MB/s	1+ GB/s

Table 2. IBM 4764 vs. iP4@3.4Ghz/OpenSSL 0.9.7f)

here is to understand how much faster a signature of x bits is, given as known baseline the time taken by an n bit signature. This is not difficult, however space constraints prevent further elaboration.

HMACs. We note that an even faster alternative is to replace short-lived signatures with simple and fast keyed message authentication codes (e.g., HMACs). This would in effect remove any authentication bottlenecks during burst periods, thus allowing practically unlimited throughputs at levels only restricted by the SCPU – main memory bus speeds (e.g., $100 - 1000MB/s$). The only drawback of this method is the inability of clients to verify any of the HMACed committed records until they are (later) signed by the SCPU. We believe that in a production environment such HMACs will be the prevalent design choice.

5 Evaluation

The introduced architecture naturally satisfies important WORM assurances: data integrity and non-repudiation.

Theorem 1. *Data records committed to WORM storage can not be altered or removed undetected.*

Theorem 2. *Insiders with super-user powers are unable to "hide" active data records from querying clients by claiming they have expired or were not stored in the first place.*

Performance. We consider a setup consisting of an unsecured main CPU (P4 @ 3.4 Ghz) and the IBM 4764-001 PCI-X Cryptographic Coprocessor [2] (see table 2).

Figure 1 illustrates the system throughput with increasing record sizes. By deploying the various deferred strong constructs optimization (section 4.3, with 512 bit signatures for the weak constructs), update rates of over 2000 - 2500 records/second are possible in burst of no more than 60-180 minutes (life-time of the short-lived constructs). Without deferring strong constructs, the WORM layer can support sustained throughputs of 450-500 records/second. These results naturally scale if multiple SCPUs are available.

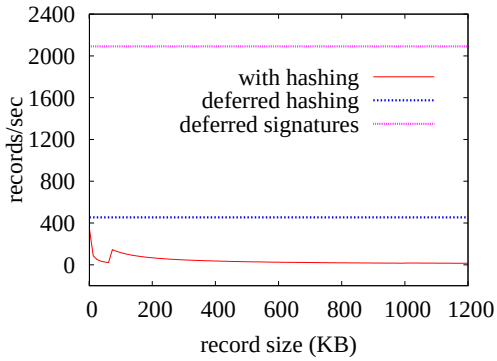

Figure 1. Throughput variation with records size. Deferring the signatures yields a throughput of 2000-2500 records/s.

Ultimately, it is likely that even for single-CPU (but especially for multi-CPU) systems, I/O seek and transfer overheads are likely to constitute the main operational bottlenecks (and not the WORM layer). Typical high-speed enterprise disks feature 3-4ms+ latencies for individual block disk access, twice the projected average SCPU overheads – these can become dominant, especially when considering fragmentation and entire multi-block file accesses.

6 Conclusions

Recent compliance regulations are intended to foster and restore humans' trust in digital information records and, more broadly, in our businesses, hospitals, and educational enterprises. As increasing amounts of information are created and live digitally, compliance storage will be a vital tool in restoring this trust and ferreting out corruption and data abuse at all levels of society.

In this paper we first identified essential vulnerabilities in existing regulatory compliance systems. We outlined the requirement for strong mechanisms. We introduced a regulatory-compliant architecture offering strong Write Once Read Many assurances by leveraging tamper-proof hardware. We have shown the architecture to handle throughputs of over 2500 transactions/second.

In future research it is important to explore traditional file system primitives layered on top of block-level WORM.

Acknowledgments. We would like to thank Simona Boboila, Xiaonan Ma, Ron Perez, Marianne Winslett, and our anonymous reviewers.

Complete Version. To fit the allotted space, this paper has been shortened considerably. A complete version of this paper will be at `www.cs.stonybrook.edu/~sion`.

References

[1] IBM Common Cryptographic Architecture (CCA) API. Online at http://www-03.ibm.com/security/cryptocards//pcixcc/overcca.shtml, 2006.

[2] IBM 4764 PCI-X Cryptographic Coprocessor. Online at http://www-03.ibm.com/security/cryptocards/pcixcc/overview.shtml, 2007.

[3] Trusted Platform Module (TPM) Specifications. Online at https://www.trustedcomputinggroup.org/specs/TPM, 2007.

[4] M. Bellare and D. Micciancio. A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost. In W. Fumy, editor, *Advances in Cryptology — Proceedings of EuroCrypt*, volume 1233 of *Lecture Notes in Computer Science*, pages 163–192. Springer-Verlag, 11–15 May 1997.

[5] M. Blaze. A Cryptographic File System for Unix. In *Proceedings of the first ACM Conference on Computer and Communications Security*, pages 9–16, Fairfax, VA, 1993. ACM.

[6] D. E. Clarke, S. Devadas, M. van Dijk, B. Gassend, and G. E. Suh. Incremental multiset hash functions and their application to memory integrity checking. In C.-S. Laih, editor, *ASIACRYPT*, volume 2894 of *Lecture Notes in Computer Science*, pages 188–207. Springer, 2003.

[7] EMC. Centera Compliance Edition Plus. Online at http://www.emc.com/centera/ and http://www.mosaictech.com/pdf_docs/emc/centera.pdf, 2007.

[8] W. Geiselmann and R. Steinwandt. Special Purpose Hardware in Cryptanalysis: The Case of 1024-bit RSA. *IEEE Security and Privacy*, pages 63–66, Jan. 2007.

[9] S. Ghemawat, H. Gobioff, and S. T. Leung. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pages 29–43, Bolton Landing, NY, October 2003. ACM SIGOPS.

[10] O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.

[11] P. C. Gutmann. Secure filesystem (SFS) for DOS/Windows. www.cs.auckland.ac.nz/~pgut001/sfs/index.html, 1994.

[12] R. Hasan, R. Sion, and M. Winslett. Introducing Secure Provenance. In *StorageSS*, 2007. Stony Brook Network Security and Applied Cryptography Lab TR 03-2007.

[13] L. Huang, W. W. Hsu, and F. Zheng. CIS: Content Immutable Storage for Trustworthy Record Keeping. In *Proceedings of the Conference on Mass Storage Systems and Technologies (MSST)*, 2006.

[14] IBM Corp. IBM TotalStorage Enterprise. Online at http://www-03.ibm.com/servers/storage/, 2007.

[15] A. Kashyap, S. Patil, G. Sivathanu, and E. Zadok. I3FS: An In-Kernel Integrity Checker and Intrusion Detection File System. In *Proceedings of the 18th USENIX Large Installation System Administration Conference (LISA 2004)*, pages 69–79, Atlanta, GA, November 2004. USENIX Association.

[16] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.

[17] R. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Research in Security and Privacy*, 1980.

[18] Microsoft Research. Encrypting File System for Windows 2000. Technical report, Microsoft Corporation, July 1999. www.microsoft.com/windows2000/techinfo/howitworks/security/encrypt.asp.

[19] National Association of Insurance Commissioners. Graham-Leach-Bliley Act, 1999. www.naic.org/GLBA.

[20] Z. Peterson, R. Burns, G. Ateniese, and S. Bono. Design and Implementation of Verifiable Audit Trails for a Versioning File System. In *Proceedings of the Conference on File and Storage Technolgoies (FAST), USENIX* , pages 93–106, 2007.

[21] C. Pomerance. A Tale of Two Sieves. *Notices of the ACM*, pages 1473–1485, Dec. 1996.

[22] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *Proceedings of the First USENIX Conference on File and Storage Technologies (FAST 2002)*, pages 89–101, Monterey, CA, January 2002. USENIX Association.

[23] D. Santry, M. Feeley, N. Hutchinson, and A. Veitch. Elephant: The file system that never forgets. In *Workshop on Hot Topics in Operating Systems*, pages 2–7, 1999.

[24] R. D. Silverman. A cost-based security analysis of symmetric and asymmetric key lengths. Online at http://www.rsasecurity.com/rsalabs/bulletins/index.html. Note: Bulletin 13, 2000.

[25] The Enterprise Storage Group. Compliance: The effect on information management and the storage industry. Online at http://www.enterprisestoragegroup.com/, 2003.

[26] The U.S. Department of Defense. Directive 5015.2: DOD Records Management Program. Online at http://www.dtic.mil/whs/directives/corres/pdf/50152std_061902/p50152s.pdf, 2002.

[27] The U.S. Department of Education. 20 U.S.C. 1232g; 34 CFR Part 99:Family Educational Rights and Privacy Act (FERPA). Online at http://www.ed.gov/policy/gen/guid/fpco/ferpa, 1974.

[28] The U.S. Department of Health and Human Services Food and Drug Administration. 21 CFR Part 11: Electronic Records and Signature Regulations. Online at http://www.fda.gov/ora/compliance_ref/part11/FRs/background/pt11finr.pdf, 1997.

[29] The U.S. Securities and Exchange Commission. Rule 17a-3&4, 17 CFR Part 240: Electronic Storage of Broker-Dealer Records. Online at http://edocket.access.gpo.gov/, 2003.

[30] U.S. Dept. of Health & Human Services. The Health Insurance Portability and Accountability Act (HIPAA), 1996. www.cms.gov/hipaa.

[31] U.S. Public Law 107-347. The E-Government Act, 2002.

[32] U.S. Public Law No. 107-204, 116 Stat. 745. Public Company Accounting Reform and Investor Protection Act, 2002.

[33] C. P. Wright, M. Martino, and E. Zadok. NCryptfs: A Secure and Convenient Cryptographic File System. In *Proceedings of the Annual USENIX Technical Conference*, pages 197–210, San Antonio, TX, June 2003. USENIX Association.